


- (0) Build two C++ classes (and the appropriate .h declaration files and .cpp implementation files): a base class named **Employee** and a derived class called **Faculty**. In inheritance-speak: a **Faculty** (object) “isa” **Employee**.

Employee		
<i>Behaviors (public)</i>		<i>Data</i>
- CTOR (default, parameterized, and copy)		name (string)
- DTOR		age (integer)
- greeting(string aPerson): <i>object greets aPerson</i>		
- farewell(void)		
- “setters”: setName(string), setAge(integer)		
- “getters”: getName(), getAge()		
		
		“isa” (inherits from)
Faculty: public Employee		
<i>Behaviors (public)</i>		<i>Data</i>
- CTOR (default, parameterized, and copy)		<i>inherit name</i>
- DTOR		<i>inherit age</i>
		hasTenure (boolean)
- (extend) greeting(), where “extend” means to use <i>Employee</i> greeting first, then add on an additional faculty-centric greeting		
- (override) farewell(), where “override” means to use different farewell message		
- setters: setTenure(boolean)		
- getters: doesHaveTenure()		

- (1) First, let’s build a basic (no-frills) version of these objects. Download a test-driver `main.cpp` from onCourse. Your classes must work with this file. You’ll notice that I have commented out statements in `main()`. This is to show you *by example* how I worked on the objects. I don’t try to get everything working at once. Rather, I first get my basic CTORs working (Part 1). Once that works, I uncomment other lines and work on one method at a time.

// Part 1

- In *every* method of both classes, you must use the implicit **this->** pointer. This is just a good C++ programming style/habit to use.
- In your default CTORs, use bogus strings for names (e.g., “NO NAME GIVEN”) and a bogus age (e.g., -1). This is a good way to catch an object that was created, but never set.
- In each CTOR and DTOR, include a print statement. This will enable you to visually “see” when an object’s CTOR or DTOR is implemented (that is, when an object is created and when that object is destroyed or leaves scope). Give the object’s name in the “created” and “destroyed” messages (see sample output below).
- In the `greeting()` and `farewell()` methods, always state the object’s name that is doing the talking (see sample output below).

```

int main(void)
{
    // PART 1
    cout << "----- starting main()'s scope -----" << endl;

    Employee E;

    Employee Jane("Jane", 21);

    Faculty F;

    Faculty Pete("Pete", 44, false);

    // ----- end Part 1 -----

    cout << endl << "----- leaving main()'s scope -----" << endl;
    return 0;
}

```

Expected OUTPUT for Part 1

```

----- starting main()'s scope -----
Employee (default) CREATED object: [NO NAME GIVEN,-1]
Employee (param)   CREATED object: [Jane,21]
Employee (default) CREATED object: [NO NAME GIVEN,-1]
Faculty  (default) CREATED object: [NO NAME GIVEN,-1,0]
Employee (param)   CREATED object: [Pete,44]
Faculty  (param)   CREATED object: [Pete,44,0]

----- leaving main()'s scope -----
Pete Faculty object KILLED
Pete Employee object KILLED ...
NO NAME GIVEN Faculty object KILLED
NO NAME GIVEN Employee object KILLED ...
Jane Employee object KILLED ...
NO NAME GIVEN Employee object KILLED ...

```

Call us over when you get the entire main () to work for Part 1.

- (2) Uncomment /* Part 2 ... */. Implement the farewell() and greeting() methods. Remember, the Faculty greeting should first “say” the (base class) Employee greeting and then “extend” that with an additional message. However, the Faculty farewell should be completely different. The output for Part 2 is shown below.

```

----- Start PART 2 -----
Jane (Employee) says: WELCOME Foobar
Pete (Employee) says: WELCOME Mary
Pete (Faculty)  says: Morning scholar Mary
Jane (Employee) says: Bye bye ...
Pete (Faculty)  says: See you NEXT class ...

```

Call us over when you get the entire main () to work for Part 2.

- (3) Uncomment `/* Part 3 ... */`. Implement all the “setters” and “getters” in each class as needed.

```
----- Start PART 3 -----
Janey is this old: 22
Petey is this old: 45 and he does have tenure.
```

- (4) Uncomment `/* Part 4 ... */`, but do so one line (or so) at a time. Implement the needed functionality.

```
----- Start PART 4 -----
Employee (param) CREATED object: [Pete,44]
Faculty (copy CTOR) CREATED object: [Pete,44,0]
-----
Name:    Pete
Age:     44
Tenure:  does NOT have tenure.
-----

Employee (param) CREATED object: [Dr. Briefly,99]
Faculty (param) CREATED object: [Dr. Briefly,99,1]
-----
Name:    Dr. Briefly
Age:     99
Tenure:  does have tenure.
-----

Dr. Briefly Faculty object KILLED
Dr. Briefly Employee object KILLED ...
Employee (default) CREATED object: [NO NAME GIVEN,-1]
Faculty (default) CREATED object: [NO NAME GIVEN,-1,0]
-----
Name:    JoeSmo
Age:     44
Tenure:  does have tenure.
-----
```